Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search

Oleg Savenko[1], Sergii Lysenko[2], Andrii Nicheporuk[3] and Bohdan Savenko[4]

Khmelnitsky National University, Khmelnitsky, Ukraine
[1]savenko_oleg_st@ukr.net
[3]sirogyk@ukr.net
[3]andrey.nicheporuk@gmail.com
[4] savenko_bohdan@ukr.net
ki.khnu.km.ua

Abstract. The article presents a new technique for metamorphic viruses detection based on the search of equivalent functional blocks. The method takes into account the obfuscation techniques of blocks reordering.
The method involves the searching of the correspondences between the functional blocks of the metamorphic versions, and consists of two stages. On the first stage the equivalent functional blocks based on the statistical evaluation of the instructions appearance in the block are to be searched. The second stage involves the choice refinement of equivalent blocks and selection the most appropriate block, which will be used for the the forming of the feature vector of similarity for metamorphic viruses' versions. The method carries out the classification of feature vectors with the involvement of fuzzy logic. The proposed method allows to reduce the number of false positives in comparison with the previous study.

Keywords: metamorphic viruses, functional block, basic blocks, obfuscation, opcode.

Key Terms: Model-Based Software System Develoment, SoftwareComponent, Software System.

1 Introduction

Today in the world the problem of the virus detection is very actual. The number of new malware is growing up rapidly. In particular, according to Symantec in the second half of 2016, there were about 96 million unique copies of the malicious software [1]. The main profit concerning to the virus spreading is the stealing of confidential information, damage the operation of computer systems, own ambitious motives, etc.

Among all set of virus programs the metamorphic viruses occupied one of the leading places. According to the Kaspersky company the metamorphic virus Virus.Win32. Sality.gen is in the top five of the most spread viral threats (5.53% of all local threats) [2]. The main difficulty of metamorphic viruses detection is due to usage of the techniques of reordering and replacement of its own instructions. Each new version created by the metamorphic virus varies from the existing ones. This feature downplays the signature analysis usage, which is the base of most antivirus tools [3].

This paper is devoted to solving the problem of the metamorphic viruses detection, where the similarities between its modified versions is more than 10%. In particular, researches presented in [4] have demonstrated that metamorphic versions' similarity at about 10% is characteristic for NGVCK metamorphic generators. Versions of code, generated by this tool, are considered to be one of the most obfuscated. Other classes of metamorphic viruses in the work are not considered because they are unapplicable and have a large computational complexity for the development and detection [5].

2 Related Works

Research community pay particulat attention to the problem of the metamotphic viruses spread [6-10], however, the effectiveness of detection techniques is still unsufficient.

In [6] authors involved the markov chain of instruction trace to make graph kernel and made similarity matrix based on transition probability between instructions. The classification is made by using the support vector machine. Approach is based on the usage of Ether malware analysis framework based on Xen Virtual machine for execution of binary. It is able to identify more then 100 basic instructions by the monitoring procedure. It also is ble to execute the similarity check, which is based on the usage of the Guassian and Eigen vector method. This approach showed efficiency of detection at the level of 96.41%, including polymorphic viruses, which is significantly higher than the known antiviral tools, however, the authors didn't take into account metamorphic viruses, which in many cases are similar to polimorphic.

In [7] the aproach for metamorphic malware detection is presented. It is based on the evaluation of the the similarity of executables using the opcode graphs. Technique involved the opcodes extraction from the program, and a weighted opcode graph construction. As a node of the the graph is opcode and there is an edge from the node to a successor opcode. The edge is given a weight. It takes into accoutn the frequency of opcode occurrence. Proposed approach perform the comparison of the obtained graph with the known malware graph. This comparison is based on a scoring function presented in the paper. However, the executable file size increasing leads to the increasing of the opcodes number and to the increasing of the graph size. In this case, the task can become the NP-complete.

In the [8] method for metamorphic viruses which is based on machine learning approach like support vector machine with histogram intersection kernel is proposed. It involves such steps: the extraction of the feature histograms from each portable executable file, mapping them into the feature space using a histogram intersection kernel. Using the histogram intersection kernel maked it possible to find the optimal

hyperplane for separating the metamorphic variants from benign programs in a feature space of very high dimension.

In [9], metamorphic detection was carried out using a similarity index technique based on edit distance and pairwise sequence alignment. The edit distance between two opcode sequences extracted from files is computed by replacing each opcode with a corresponding symbol. Authors test these similarity measures on the challenging problem of metamorphic virus detection. The results from the edit distance and pairwise sequence alignment methods shows that the morphed viruses having random percentages of dead code and subroutine insertions (i.e., 5%, 15%, 25% and 30%) are still detectable within a certain error rate. However the approach does not consider the use of antiemulation technology that can use viruses.

In [10], to detect metamorphic virus variants, authors presented an approach based the use of hidden Markov models (HMMs) to capture the statistical properties of viruses in the same family. They generated 200 NGVCK viruses, trained 25 models and used the trained models to classify 65 programs including both NGVCK viruses and other random non-viral programs. In most cases, presented models were able to have a detection rate of over 90% and a false positive rate of less than 10%. However, if the benign software's fragment of code is inserted into the metamorphic virus's body, approach will demonstrate the increase of false positives.

The work [11] is based on the similarity matching techniques by mean of a statistical scanner employing feature-ranking methods. Approach investigated the feature-ranking methods such as Term Frequency – Inverse Document Frequency (TF-IDF), Term Frequency – Inverse Document Frequency – Class Frequency (TF-IDF-CF), Categorical Proportional Distance (CPD), Galavotti – Sebastian - Simi Coefficient (GSS), Weight of Evidence of Text (WET), Term Significance (TS), Odds Ratio (OR), Weighted Odds Ratio (WOR), Multi Class Odds Ratio (MOR), Comprehensive Measurement Feature Selection (CMFS), and Accuracy2 (ACC2) as the base of metamorphic viruses detection. The classification of malware and benign programs is performed by considering top ranked features obtained using individual feature selection methods. In order to ascertain applicability in real time malware scanner, evaluation of feature ranking methods, were performed using McNemar test.

However, the proposed approaches based on statistical evaluation of instruction are uneffective for metamorphic viruses, which are using the technique of the code's blocks replacement, because the frequency of the instructions occurrence in a modified version of metamorphic virus will not be changed.

A state of art demonstrates the necessity of the development of the new approaches for metamorphic viruses' detection, which will be able to improve its efficiency.

3 Previous Work

In [12]presented a technique for metamorphic virus's detection, based on the usage of the modified emulators in the corporate area network. In the proposed approach any program that comes from the Internet to the host is checked by the suspicion program analyzer and is sent to every host on the network. If the file is defined

as suspicious, it goes to the emulation unit in order to obtain the modified versions of the same file. On the next stage the comparison of the original version of file before the emulation with the modified file's version aftter emulation is performed. In order to compare two versions of program, it is partitioned into functional blocks and the comparison is performed using the Damerau-Levenshtein metrics. The result of the comparison process is the feature vector of the similarity for the metamorphic viruses' versions. In order to provoke the metamorphic properties of program, each host of the network was equipped with a modified emulator, which had different conditions for suspicious code execution.

The similarity vectors for the versions of the metamorphic viruses, obtained from each host of the network, are sent to the server, where the conclusion about the membership of suspicious program to one of the metamorphic viruses' classes is made. If such program is identified as a virus, information about it is sent to the host, which was infected by program, and the program is blocked.

Experimental results presented in [12] demonstrated the efficiency of the metamorphic viruses' detection at the level of 85%. However, the proposed technique showed a great number of false positives. The main reason was that the functional blocks which were compared in order to obtain the feature vector of the similarity for the metamorphic viruses' versions in many cases were not equivalent.

4 Metamorphic Viruses' Detection Technique Based on the Equivalent Functional Block Search

In order to eliminate the disadvantages of the technique described in [12] and increase its efficiency, a new approach for unknown metamorphic viruses' detection is proposed. It includes improvements in the term of the functional blocks choice for its comparison, which will reduce the rate of false positives and increase the efficienty of detection.

The procedure of the equivalent functional blocks search for comparison consists of two steps. At the first stage, the equivalent functional blocks are determined. Such determination is based on the statistical evaluation of the instructions appearance in the block. The second stage involves the choice refinement of equivalent blocks and selection the most appropriate block, which will be used for the purpose of the rating evaluation of similarity between the program F_p before emulation and the program F_s after the emulation.

Let us assume the functional block FB as the maximal sequence of disassembled instructions $\{I_1, I_2, \ldots, I_m, I_j\}$ that is characterized by the following properties:

- the control flow must enter the block from the first instruction;
- the block must not contain the instructions of unconditional or conditional jumps;
- the end of the block must have at most one control-flow instruction.

For automatic generation of functional blocks that meets such properties, the IDA Pro disassembler with Gaph view option was used. In order to simplify the analyzing and processing procedures, the operands of the instruction are ignored.

Let us describe a program F as a directed graph. Let us denote V – a set of functional blocks of program F, that is $V = \{FB_1, FB_2, ..., FB_n\}$. Thus $E \subseteq V \times V \times \{True, False\}$ is the jump in the the control flow between the blocks, caused by the control transfer instructions, where *True* and *False* specify the conditions of the jump, then $F = \{V, E\}$ it will be a directed graph, where the nodes are functional blocks, and the edges – connections between the blocks in the control flow of the program.

4.1 Search of the Equivalent Functional Blocks

In order to avoid detection by antiviruses the metamorphic viruses use a wide range of evasion techniques, such as garbage instructions insertion (junk code), blocks reordering, usag of the equivalent instructions and registers [6-8]. The usage of these techniques allows creation of metamorphic versions with the same functionality, but using different instructions (table 1). It limits the advantages of the signature method.

Table 1. The usage of obfuscation techniques: junk code insertion, blocks reordering and instruction replacement

Original code	Junk code insertion	Block reordering	Instruction replacemnt

call 0h
pop ebx
lea ecx, [ebx+42h]
push ecx
push eax
push eax
sidt [esp-05h]
pop ebx
add ebx,1Ch
cli
mov ebp, [ebx]
	call 0h
	pop ebx
	lea ecx, [ebx+42h]
	nop
	xor ax,ax
	push ecx
	push eax
	inc eax
	push eax
	dec [esp-0h]
	dec eax
	sidt [esp-02h]
	pop ebx
	add ebx,1Ch
	cli
	mov ebp, [ebx]
		call 0h
		pop ebx
		jmp S2
		S3: push eax
		push eax
		sidt [esp-02h]
		jmp S4
		add ebx, 1Ch
		jmp S6
		S2: lea ecx,
		[ebx+42h]
		push ecx
		jmp S3
		S4: pop ebx
		cli
		jmp S5
		S5: mov ebp, [ebx]
			call 0h
			pop ebx
			lea ecx, [ebx+42h]
			sub esp, 03h
			sidt [esp-02h]
			add [esp], 1Ch
			mov ebx, [esp]
			inc esp
			cli
			mov ebp, [ebx]

In order to decrease the computational complexity method involves no processing all executable but only single PE EXE section.

Thus, on the step of determining the similarity of a suspicious program to the metamorphic virus, on the basis of search of the equivalent functional blocks, an important task is the localization of search. Because constituent units in the structure of

executable files of the PE EXE format is sections, search of the equivalent functional blocks will be carried out only in certain sections.

Selection of sections, in which should be searched equivalent functional blocks between the programs before and after the emulation is carried out according to the following rules:

Determining the entry point of the program and section in which it located.

- if the name of this section differs from the standard names of the sections (.text .data, etc) or the section has the attribute of the access as executable, then the section is defined as a *labeled section for comparison*;

- if in a section in which is located the entry point, has a call or a jump that contains the address of the last section, then the section is defined as a *labeled section for comparison*;

- else last section of executable is defined as a *labeled section for comparison*.

After determining of PE EXE sections for the program before emulation and appropriate section for the program after emulation, the next step is to search of the equivalent functional blocks between these programs.

Let us assume the equivalent functional blocks of the programs A and B two or more functional blocks, which perform the same functions and are modified using the code obfuscation.

Let us denote the program before emulate as F_p, and after emulation – F_s. After the disassembly, performed by the interactive disassembler IDA Pro, two sets of functional blocks are obtained: $FB^{F_p} = \{fb_1^{F_p}, fb_2^{F_p}, ..., fb_m^{F_p}\}$ and $FB^{F_s} = \{fb_1^{F_s}, fb_2^{F_s}, ..., fb_n^{F_s}\}$. Then in order to find the equivalent functional blocks the Term Frequency – Inverse Document Frequency statistical metric applied for each function block of programs F_p and F_s, is used:

$$s_{FB} = \frac{n_i}{\sum_k n_i} * \log(\frac{N + 1.0}{n_j}) \quad (1)$$

where, n_i - the number of occurrences of the *i*-th opcode into the functional block;

$k = \overline{1, k_a}$ – the number of opcode in functional block, where k_a – total number of the assembler instructions;

N – total number of function blocks, $N_{F_p} \neq N_{F_s}$;

n_j – the number of functional blocks where the *i*-th opcode is placed.

The result of the statistical evaluation of the presence opcode in FB for program before emulation F_p and for the program after emulation F_s are the rating matrices $M(FB^{F_p})$ and $M(FB^{F_s})$. The rows of matrix contain the functional blocks of the program, and columns – the opcodes presented in the function block. Each cell of the matrix determines the appearance score of the *i*-th opcode in the *j*-th functional block (fig. 1):

$M(FB^{F_p}) =$

	i_1	i_2	…	i_k
$FB_1^{F_p}$	s_{11}	s_{12}	…	s_{1k}
$FB_2^{F_p}$	s_{21}	s_{22}	…	s_{2k}
…	…	…	…	…
$FB_m^{F_p}$	s_{m1}	s_{m2}	…	s_{mk}

a)

$M(FB^{F_s}) =$

	i_1	i_2	…	i_g
$FB_1^{F_s}$	s_{11}	s_{12}	…	s_{1g}
$FB_2^{F_s}$	s_{21}	s_{22}	…	s_{2g}
…	…	…	…	…
$FB_n^{F_s}$	s_{n1}	s_{n2}	…	s_{ng}

b)

Fig.1. Rating matrix of the opcodes appearance in function bloks for the program: a) before emulation $M(FB^{F_p})$; b) after emulation $M(FB^{F_s})$

In order to evaluate the equivalent functional blocks, the next step requires the calculation of the similarity score between two functional blocks of the program F_p and F_s. For this purpose, the squared Euclidean metrics was used:

$$E\,(FB_i^{F_p}, FB_j^{F_s}) = \sum_{i=0,\, j=0}^{k} (s_i - s_j)^2, \tag{2}$$

where, s_i – evaluation of the opcode appearance in the *i*-th block for program F_p, s_j – evaluation of the opcode appearance in *j*-th block of program F_s.

If the value of similarity score between two functional blocks is less the defined threshold δ, $E\,(FB_i^{F_p}, FB_j^{F_s}) \le \delta$, then the recalculatio of similarity score between the functional blok of the program $FB_i^{F_p}$ and the next block that follows the block $FB_j^{F_s}$, $E(FB_i^{F_p}, FB_j^{F_s} + FB_{j+1}^{F_s})$, is performed. Mentioned above steps are repeated until the value of the evaluation of the similarity is less than or equal to the threshold δ. Threshold value is defined in experimental way.

It is possible that functional block of the program F_p may correspond to several functional blocks of the program F_s (Fig. 2). The reason is that the metamorphic virus may apply the technology of the code partitioning of its code into blocks.

An example of a schematic presentation of the equivalent program's functional block before and after emulation placed in the two-dimensional space is shown in Fig. 2. For example, one block of program before emulation, can correspond to 5 equivalent functional blocks of the program after the emulation. In ordr to to eliminate the uncertainty, it is necessary to carry out the choice refinement of equivalent functional blocks.

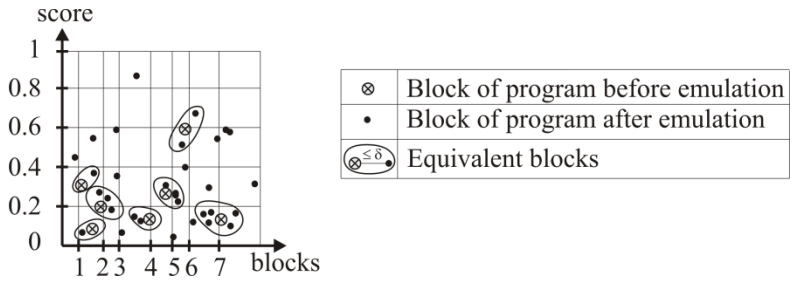


Fig. 2. A schematic presentation of the equivalent program's functional block before and after emulation placed in the two-dimensional space

4.2 The Choice Refinement of Equivalent Functional Blocks

The task of the choice refinement of equivalent blocks is to selection the most appropriate block, obtained in the previous step. For this purpose, the minimum value of the similarity among the set of equivalent functional blocks is chosen:

$$FB_i^{F_p} \equiv \min(eFB_1^{F_s}, eFB_2^{F_s}, \ldots, eFB_n^{F_s}), \tag{3}$$

where, $eFB_1^{F_s}, eFB_2^{F_s}, \ldots, eFB_n^{F_s}$ – equivalent functional blocks corresponding to the block $FB_i^{F_p}$.

In order to perform the choice refinement of equivalent blocks let us define the probability of the opcodes sequence in the functional block. For this purpose for each equivalent functional blocks $eFB_1^{F_s}, eFB_2^{F_s}, \ldots, eFB_n^{F_s}$ and block $FB_i^{F_p}$ let us construct a probability matrix for the opcodes sequence. Each cell of the matrix will consist the ratio of the number of the opcodes pair appearance to the total number the opcodes in the row.

For example, if functional block is defined by the following opcodes sequence: *mov, push, lea, pop, mov, push, push, push, call,mov*, then the probability matrix for opcodes sequence would be as shown in fig 3.

	mov	push	lea	pop	Call
mov	0	0	0	1	1
push	2	2	0	0	0
lea	0	1	0	0	0
pop	0	0	1	0	0
call	0	1	0	0	0

	mov	push	lea	pop	call
mov	0	0	0	1/2	1/2
push	2/4	2/4	0	0	0
lea	0	1	0	0	0
pop	0	0	1	0	0
call	0	1	0	0	0

Fig. 3. Probability matrix for opcodes sequence in the functional block

The last step of the equivalent functional blocks' determination is comparing of the probability matrices of opcodes sequence for the program before and after emulation (4) and choice of the minimum similarity:

$$R = \frac{1}{N^2}\left(\sum_{i,j=1}^{N-1} | a_{i,j} - b_{i,j} |\right)^2, \quad (4)$$

where, $a_{i,j}$ the matrix cell for the functional block FB^{Fp}, $b_{i,j}$ – the matrix cell for the functional block eFB^{Fs}, N – total number of opcodes for the pairs of blocks.

The obtained estimate for pairs of blocks allows determining the equivalent functional blocks with high probability.

4.3 Biulding the Feature Vector of Similarity for Metamorphic Viruses' Versions

After receiving the pairs of the equivalent functional blocks, the next step is to pairwise compare them using Damerau-Levenshtein metrics and to construct the feature vectors of the metamorphic viruses' samples' similarity using the algorithm of Wagner-Fisher.

Let us present the the feature vectors of the metamorphic viruses' samples' similarity as a tuple:

$$\overline{V_m} = \left\langle \begin{pmatrix} L(\varepsilon_1), X(\varepsilon_1), D(\varepsilon_1), I(\varepsilon_1), R(\varepsilon_1), M(\varepsilon_1) \\ L(\varepsilon_n), X(\varepsilon_n), D(\varepsilon_n), I(\varepsilon_n), R(\varepsilon_n), M(\varepsilon_n) \end{pmatrix}, Y \right\rangle, \quad (4)$$

where $\varepsilon_1,...,\varepsilon_n$ pairs of the equivalent functional blocks between the program before and after the emulation, n – a number of the equivalent blocks; L – the Damerau-Levenshtein distance between the equivalent blocks ε_i of the program before and after emulation; X – the number of the required opcode exchange operations; D – the number of the required opcode removal operations; I – the number of the required opcode insertion operations; R – the number of the required opcode replacement operations; M – the number of matches between opcodes in the equivalent functional blocks of the program before and after emulation; Y – the danger degree of the program's behavior.

The danger degree of the program's behavior is estimated on the basis of the analysis of API calls that describe the potentially dangerous behavior of the metamorphic virus. Let us present the behavior of the known metamorphic virus as a pattern (as bit strings) $P = \phi_1,...,\phi_n\zeta_1,...,\zeta_k\psi_1,...,\psi_l\iota_1,...,\iota_b\pi_1,...,\pi_u\xi_1,...,\xi_r$, where $\Phi = \{\phi_n\}_{n=0}^{K\phi}$ – a set of file functions; $Z = \{\zeta_k\}_{k=0}^{K_\zeta}$ – a set of API functions that check whether the program execution is performed in the virtual environment; $\Psi = \{\psi_l\}_{l=0}^{K_\psi}$ – a set of functions needed to implement the installation of the new components to the system; $\mathrm{I} = \{\iota_b\}_{b=0}^{K_\iota}$ – a set of functions that provide access to the Internet; $\Pi = \{\pi_u\}_{u=0}^{K_\pi}$ – a set of processes' and threads' functions; $\Xi = \{\xi_r\}_{r=0}^{K_\xi}$ – a set of API calls for infor-

mation system definition; $K_\phi, K_\zeta, K_\psi, K_\iota, K_\pi, K_\xi$ – a number of the corresponding to API calls; f - function of the destructive commands execution, which demonstrates the construction of the pattern for the metamorphic virus behavior P, $\Phi \times Z \times \Psi \times I \times \Pi \times \Xi \xrightarrow{f} P$.

Thus, the monitored suspicious program's behavior we can present as the string $R = a_1,\ldots,a_p$, where a_i – a sequence of the API-calls of the observed program.

Let us assume the boolean function of the string matching $\vartheta = (P, R)$ between the known behavior pattern and the behavior of the observed program which indicates matching or mismatching.

Let us divide the set of behavior patterns $P_Y = \{P_{Y_{high}}, P_{Y_{medium}}, P_{Y_{low}}\}$ into the three groups, which indicate the suspicious degree: high, medium and low.

Each group contains a set of patterns that describe the fullness of viruses' lifecycle implementation. The example of patterns that belong to three different groups are given below:

Y_{high} : GetSystemDirectoryA → FindFirstFileA → OpenProcess → Socket → Connect

Y_{medium}: GetSystemDirectoryA → FindFirstFileA → OpenProcess

Y_{low}: GetSystemDirectoryA → FindFirstFileA

Thus, having the behavior of the monitored program (formed pattern on the modified emulator of the host), it is to be compared with the set of with patterns of known malware. If there is a matching between the behavior of the monitored program and one malicious pattern, we are interesting in the suspicious degree *Y* of this pattern. It will be used in the procedure of the making the conclusion about the system infection with the metamorphic virus.

Note. In order to solve the string comparison problem the approximate string matching algorithm was used. It deals with the k differences problem solving. If we are given two strings, the sequence *T* = *t*1*t*2...*tm* and the *pattern* Φ=y1y2…yn in some alphabet Σ, and an integer *k,* the algorithm enables finding all substrings Φ′ of *T* with the edit distance at most *k* from Φ. The edit distance intends the minimum number operations for editing (the differences) which are required for converting Φ′ to Φ. [13]. The patterns preprocessing needs time *O*(*mn*).

4.4 Making the Conclusion About the System Infection with the Metamorphic Virus

In order to make a conclusion about the systems infection, the obtained feature vectors of the metamorphic viruses' samples' similarity from each host are to be classified by the means of the fuzzy inference system [14,15] (fig. 4).

The input linguistic variables of the FIS are: «the similarity degree of the suspicious program with its modified version based on the Lowenstein distance » (*L*), «the similarity degree of the suspicious program with its modified version based on the number of insert operations» (*I*), «The similarity degree of the suspicious program with its modified version based on the number of removal operations» (*D*), «the simi-

larity degree of the suspicious program with its modified version based on the number of replace operations» (*R*), «The similarity degree of the suspicious program with its modified version based on the number of permutation operations» (*X*) , «The similarity degree of the suspicious program with its modified version based on the number of match operations» (*M*) and «the danger degree of the program» (*Y*).

Let us assume the output linguistic variable as «the similarity degree to the metamorphic virus» (*SM*).

Each input and output linguistic variables are defined by the term set: *Low, Medium, High*. As the membership functions for inputs the trapezoid was chosen, for the output - triangular. In order to determine the program's similarity to metamorphic virus 87 rules are involed. For example, one of the rules can be presented as follows:

if (*L is Medium*) *and* (*X is High*) *and* (*D is Medium*) *and* (*I is High*) *and*

and (*R is Low*) *and* (*M is Medium*) *and* (*Y is High*) *then SM is High*

Having the result obtained by the fuzzy inference system, the suspicious program is blocked or continue its execution.

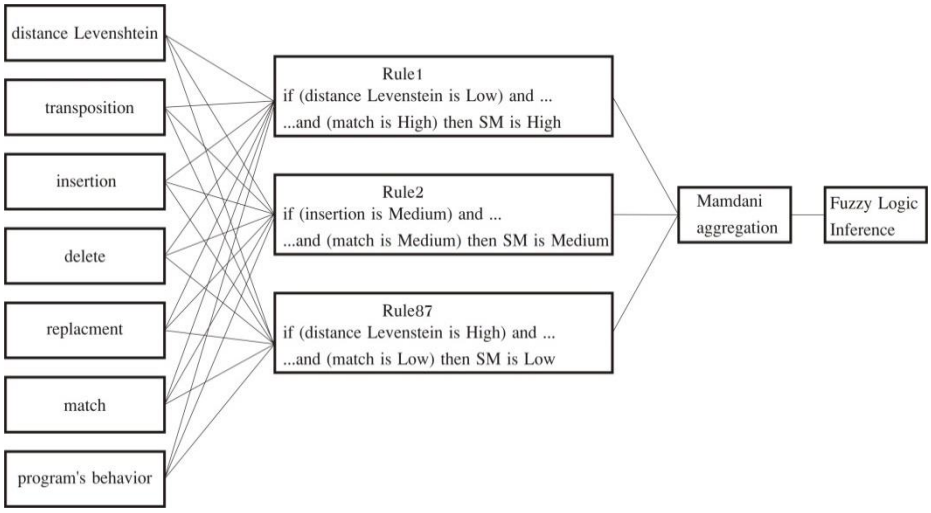


Fig. 4. The scheme of fuzzy inference system for the determination of the membership degree for each virus version to one of the metamorphic viruses' class

5 Experiments

In order to determine the efficiency of the proposed technique series of experiments was conducted. To do this, the set of metamorphic virus versions' were generated. For this purpose three types of metamorphic generators: Next Generation Virus Creation Kits (type NGVCK), Second Generation Virus Generator (type G2) and Virus Creation Lab for Win32 (type VCL32)[16] were used. Thus, the 228 programs with the features of the metamorphic viruses (76 programs of each NGVCK, VCL32 and G2 type) were generated. The set of all generated metamorphic viruses were di-

vided into two groups: one group for training set (set of prototypes) and another for testing (38 instances in each).

Each program (from both sets) was executed in the emulator (Qemu [17]) for the purpose of its new version obtaining [12]. Each program was disassembled and partitioned into the functional blocks using the interactive disassembler IDA Pro [18]. In order to choose the equivalent functional blocks of the program, a new software that allows the similarity evaluation for a pair of functional blocks for the program before and after emulation was developed.

The similarity evaluation for a pair of functional blocks for the testing's programs before and after emulation and set of behavior patterns (as discussed above in Section 4.3) are the basis of the knowledge base for fuzzy classification.

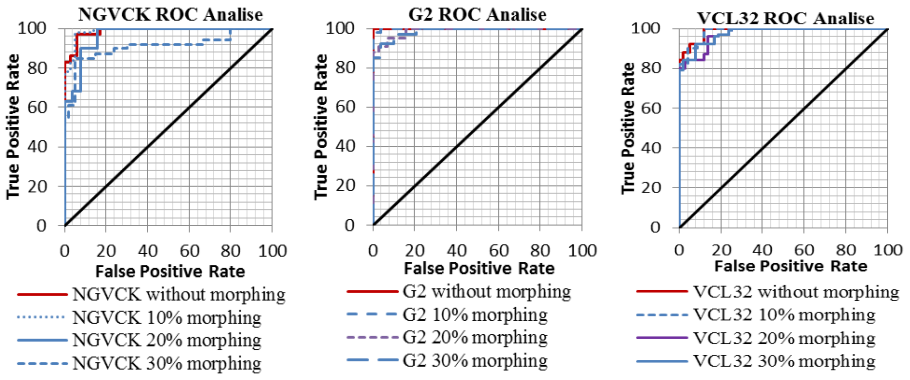
Experimnts include the investigation of the number of correctly chosen equivalent functional blocks. Table 2 shows the average size of tested programs and the results correctly chosen functional blocks of the program before and after emulation in comparison with the approach [12], described in section 3, where the block reordering is not taking into account.

Table 2. Correctly chosen functional blocks for the program before and after emulation

Metamorphic viruse's class	Number of correctly chosen FB, % (approach [12])	Number of correctly chosen FB, % (new approach)	The average programs size, bytes
NGVCK	85	96	8241
VCL32	88	100	6123
G2	91	100	2564

In order to evaluate the efficiency of the metamorpchic viruses' detection, we calculated the true positive and false positive rates. In the experiments, the value of similarity score between two functional blocks was defined as the threshold $\delta = 0.6$.

In addition, the efficiency of the proposed approach with taking into account the obfuscation degree of the generated metamorphic viruses' versions was investigated. For this purpose, each metamorphic virus was obfuscated by the insertion of the junk code – 10%, 20% and 30% of the total number of opcodes of the metamorphic virus. In Fig. 5 the ROC curves for metamorphic versions without and with obfuscation and for different values of the obfuscation degrees for NGVCK, VCL32 and G2 types of the metamorphic viruses are presented. Fig. 5 shows that a minimum level of false positives without additional obfuscation is observed in all cases (the number of false positives for G2 – 0). The highest value of false positives is observed for metamorphic versions of NGVCK with 30% of additional code obfuscation (5% false positives while 85% true positives).



a) b) c)

Fig. 5. ROC curves for metamorphic versions without and with obfuscation and with different values of the obfuscation degrees: a) NGVCK, b)VCL32, c) G2

6 Conclusions

The paper presents a new technique for metamorphic viruses detection based on the search of equivalent functional blocks. It takes into account the obfuscation techniques of blocks reordering.

The method involves the searching of the correspondences between the functional blocks of the metamorphic versions, and consists of two stages. On the first stage the equivalent functional blocks based on the statistical evaluation of the instructions appearance in the block are to be searched. The second stage involves the choice refinement of equivalent blocks and selection the most appropriate block, which will be used for the the forming of the feature vector of similarity for metamorphic viruses' versions. The method carries out the classification of feature vectors with the involvement of fuzzy logic.

The proposed technique allows metamorphic viruses detection in which the similarities between versions are more than 10%. The technique demonstrates the low level of the false positives and high level of true positives of the metamorphic viruses detection.

References

1 Security Response Publications. Monthly Threat Report. Availabe: https ://www.symantec.com/security_response/publications/monthlythreatreport.jsp

2 Kaspersky Security Bulletin 2015. Overall statistics for 2015. Availabe: https:// securelist.com/analysis/kaspersky-security-bulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015/

3 Raiyn, J.: A survey of Cyber Attack Detection Strategies. International Journal of Security and Its Applications, 8(1), pp. 247-256 (2014)

4 Desai, P., Stamp, M.: A highly metamorphic virus generator. Int. J. Multimedia Intelligence and Security, Vol. 1(4), pp. 402-427 (2010)
5 Podlovchenko, R.I., Kuzyurin, N.N., Shcherbina V.S., Zakharov V.A.: Using algebraic models of programs for detecting metamorphic malwares. Journal of Mathematical Sciences, Vol. 172 (5), pp. 740-750 (2011)
6 Anderson, B., Quist, D., Neil, J., Storlie C., Lane, T.: Graph-based malware detection using dynamic analysis. Journal in Computer Virology, 7, pp. 247-258 (2011)
7 Runwal, N., Low, R.M., Stamp, M.: Opcode Graph Similarity and Metamorphic Detection. Journal in Computer Virology, 8, pp. 37-52 (2012)
8 Nagaraju, A.: Metamorphic malware detection using base malware identification approach. Journal Security and Communication Networks, 7, pp. 1719-1733 (2014)
9 Patel, M.: Similarity tests for metamorphic virus detection. Master's thesis, San Jose State University (2011)
10 Wong, W.: Analysis and Detection of Metamorphic Computer Viruses. Master's thesis, San Jose State University (2006)
11 Kuriakose, J., Vinod, P.: Unknown Metamorphic Malware Detection: Modelling with Fewer Relevant Features and Robust Feature Selection Techniques, IAENG International Journal of Computer Science, Vol. 42(2), p139-151 (2015)
12 Pomorova, O., Savenko, O., Lysenko, S., Nicheporuk, A.: Metamorphic Viruses Detection Technique Based on the Modified Emulators. ICT in Education, Research and Industrial Applications, Integration, Harmonization and Knowledge Transfer, Vol. 1614, Kyiv, June 2016. – PP. 375-383 (2016)
13 Tarhio, J., Ukkonen, E.: Approximate BoyerMoore String Matching. SIAM Journal on Computing. - 1993, Vol. 22, No. 2, pp. 243-260
14 Drozd, J., Drozd, A., Antoshchuk, S., Kharchenko, V.: Natural Development of the Resources in Design and Testing of the Computer Systems and their Components. In: 7th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, pp. 233--237. Berlin, Germany (2013)
15 Kondratenko, Y., Kondratenko, N.: Soft Computing Analytic Models for Increasing Efficiency of Fuzzy Information Processing in Decision Support Systems. Chapter in book: Decision Making: Processes, Behavioral Influences and Role in Business Management, R. Hudson (Ed.), Nova Science Publishers, New York, 41-78 (2015)
16 VX Heavens Computer virus collection. Availabe:http://vx.netlux.org
17 Qemu. Open source processor emulator [online] Available: http://wiki.qemu.org/Main_Page
18 Hex-Rays, S.A.:IDA Pro 5.5 Hex-Rays, S.A.:IDA Pro 5.5 https://www.hex-rays.com/products/ida/